# A NATURAL DEDUCTION STYLE PROOF SYSTEM FOR PROPOSITIONAL $\mu$-CALCULUS AND ITS FORMALIZATION IN INDUCTIVE TYPE THEORIES

MARINO MICULAN

*Dipartimento di Matematica e Informatica, Università degli Studi di Udine*
*Via delle Scienze, 206, 33100, Udine, Italy.* mailto:miculan@dimi.uniud.it

In this paper, we present a formalization of Kozen's propositional modal $\mu$-calculus, in the Calculus of Inductive Constructions. We address several problematic issues, such as the use of *higher-order abstract syntax* in inductive sets in presence of recursive constructors, the encoding of modal ("proof") rules and of context sensitive grammars. The encoding can be used in the Coq system, providing an experimental computer-aided proof environment for the interactive development of error-free proofs in the $\mu$-calculus. The techniques we adopted can be readily ported to other languages and proof systems featuring similar problematic issues.

## Introduction

In this paper, we present a formalization of Kozen's propositional modal $\mu$-calculus [10], often referred to as $\mu K$, in the Coq proof assistant [4].

The $\mu$-calculus is a temporal logic which subsumes many modal and temporal logics, such as $PDL$, $CTL$, $CTL^*$, $ECTL$. Despite its expressive power, $\mu K$ enjoys nice properties such as decidability and the finite model property. The long-standing open problem of axiomatizability of $\mu K$ has been solved by Walukiewicz [19,?], who has proved the completeness of Kozen's original system given in [10]. Therefore, the $\mu$-calculus is an ideal candidate as a logic for the verification of processes. Nevertheless, like any formal systems, its applicability to non trivial cases is limited by long, difficult, error-prone proofs.

This drawback can be (partially) overcome by supplying the user with a *computer-aided proof environment*, that is, a system in which he can represent (*encode, formalize*) the formal system, more or less abstractly: its syntax, axioms, rules and inference mechanisms. After having supplied the proof environment with a representation of the formal system, the user should be able to correctly manipulate (the representations of) the proofs.

However, the implementation of a proof environment for a specific formal system is a complex, time-consuming, and daunting task. The environment should provide tools for checking previously hand-made proofs; developing interactively, step-by-step, error-free proofs from scratch; reusing previously proved properties; even, deriving properties automatically, when feasible, freeing the user from most unpleasant and error-prone steps.

An alternative, and promising solution is to develop a general theory of logical systems, that is, a *Logical Framework* (LF). A Logical Framework is a metalogical formalism for the specification of both the *syntactic* and the *deductive* notions of a wide range of formal systems. Logical Frameworks always provide suitable means for representing and deal with, in the metalogical formalism, the *proofs* and *derivations* of the object formal system. Much of the implementation effort can be expended once and for all; hence, the implementation of a Logical Framework yields a *logic-independent proof development environment*. Such an environment must be able to check validity of deductions in any formal system, after it has been provided by the specification of the system in the formalism of the Logical Framework.

In recent years, several different frameworks have been proposed, implemented and applied to many formal systems. *Type theories* have emerged as leading candidates for Logical Frameworks. Simple typed $\lambda$-calculus and minimal intuitionistic propositional logic are connected by the well-known *proposition-as-types* paradigm [5,?]. Stronger type theories, such as the *Edinburgh Logical Framework* (ELF) [9,2], the *Calculus of Inductive Constructions* (CIC) [4] and *Martin-Löf's type theory* (MLTT) [16], were especially designed, or can be fruitfully used, as a logical framework. In these frameworks, we can represent faithfully and uniformly all the relevant concepts of the inference process in a logical system: syntactic categories, terms, assertions, axiom schemata, rule schemata, tactics, etc. via the *judgements-as-types, proofs-as-$\lambda$-terms* paradigm [9]. The key concept is that of *hypothetico-general* judgement [13], which is rendered as a type of the dependent typed $\lambda$-calculus of the Logical Framework. With this interpretation, a judgement is viewed as a type whose inhabitants correspond to proof of this judgement.

It is worthwhile noticing that Logical Frameworks based on type theory directly give rise to proof systems in *Natural Deduction style* [8,17]. This follows directly from the fact that the typing systems of the underlying $\lambda$-calculi are in Natural Deduction style, and rules and proofs are faithfully represented by $\lambda$-terms. As it is well-known, Natural Deduction style systems are more suited to the practical usage, since they allow for developing proofs the way mathematicians normally reason.

These type theories have been implemented in logic-independent systems such as Coq [4], LEGO [11], and ALF [12]. These systems can be readily turned into interactive proof development environments for a specific logic: we need only to provide the specification of the formal system (the *signature*), i.e. a declaration of typed constants corresponding to the syntactic categories, term constructors, judgements, and rule schemata of the logic. It is possible to prove, informally but rigorously, that a formal system is correctly, *adequately*

represented by its specification in the Logical Framework. This proof usually exhibit bijective maps between objects of the formal system (terms, formulæ, proofs) and the corresponding $\lambda$-terms of the encoding.

In this paper, we investigate the applicability of this approach to the propositional $\mu$-calculus. Due to its expressive power, we adopt the Calculus of Inductive Constructions, implemented in the system Coq. Beside its expressive power and importance in the theory and verification of processes, the $\mu$-calculus is interesting also for its syntactic and proof theoretic peculiarities. These idiosyncrasies are mainly due to a) the negative arity of "$\mu$" (i.e., the bound variable $x$ ranges over the same syntactic class of $\mu x\varphi$); b) context-sensitive grammar (the condition on the formation of $\mu x\varphi$); c) rules with complex side conditions ("proof rules"). These anomalies escape the "standard" representation paradigm of Logical Frameworks; that is, there is no "standard" way to represent them in a Logical Framework. Hence, we will adopt new efficient representation techniques, which can be ported to other systems featuring the same anomalies. Moreover, since generated editors allow the user to reason "under assumptions", the designer of a proof editor for a given logic is urged to look for a Natural Deduction formulation which can take best advantage of the possibility of manipulating assumptions.

Beside these practical and theoretical motivations, this work can give insights in the expressive power of CIC and Coq. Indeed, the encoding techniques we will adopt take full advantage of pragmatic features offered by Coq, such as the automatic simplification of terms, in order to simplify as much as possible the task of proof development.

**Structure of this paper.** In Section 1, we recall the language and the semantics of $\mu K$. We will also introduce a semantical consequence relation, which will be the semantical counterpart of the proof system. The Natural Deduction style proof system $\mathbf{N}\mu K$ will be introduced in Section 2. In this section we will present also a proof system for capturing the well formedness condition on formulæ of the form $\mu x\varphi$. In Section 3 we will discuss the formalization of $\mu K$ in CIC. We will see that $\mu K$ arises some peculiarities which are difficulty encoded in CIC; we will present some solutions. We will suppose the reader to be familiar with the CIC and the Coq system.

Final comments and remarks are reported in Section 4. Longer listings of Coq code are reported in appendix.

## 1 Syntax, semantics and consequence relation

The language of $\mu K$ is an extension of the syntax of propositional dynamic logic. Let $Act$ be a set of *actions* (ranged over by $a, b, c$), $\Phi_0$ a set of atomic propositional letters (ranged over by $p$), and $Var$ a set of propositional vari-

ables (ranged over by $x, y, z$); then, the syntax of the $\mu$-calculus on $Act$ is:

$$\Phi: \quad \varphi ::= p \mid ff \mid \neg\varphi \mid \varphi \supset \psi \mid [a]\,\varphi \mid x \mid \mu x\varphi$$

where the formation of $\mu x.\varphi$ is subject to the *positivity condition:* every occurrence of $x$ in $\varphi$ has to appear inside an even number of negations (In the following we will spell out this condition more in detail). We call *preformulæ* the language obtained by dropping the positivity condition. The variable $x$ is *bound* in $\mu x\varphi$; the usual conventions about $\alpha$-equivalence apply. We write $\nu x\varphi$ as a shorthand for $\neg\mu x(\neg\varphi[\neg x/x])$.

The interpretation of $\mu$-calculus comes from Modal Logic. A model for the $\mu$-calculus is a transition system, that is, a pair $\mathcal{M} = \langle S, [\![\cdot]\!]_{\mathcal{M}}\rangle$ where $S$ is a (generic) nonempty set of *(abstract) states*, ranged over by $s, t, r$, and $[\![\cdot]\!]_{\mathcal{M}}$ is the interpretation of atomic propositional and command symbols: for all $p, a$, we have $[\![p]\!]_{\mathcal{M}} \subset S$ and $[\![a]\!]_{\mathcal{M}} : S \to \mathcal{P}(S)$.

Formulæ of $\mu$-calculus may have free propositional variables; therefore, we need to introduce *environments*, which are functions assigning sets of states to propositional variables: $Env \stackrel{\text{def}}{=} Var \to \mathcal{P}(S)$. Given a model $\mathcal{M} = \langle S, [\![\cdot]\!]\rangle$ and an environment $\rho$, the semantics of a formula is the set of states in which it holds, and it is defined by extending $[\![\cdot]\!]$ compositionally, as follows:

$$\begin{array}{rl} [\![p]\!]_{\mathcal{M}}\rho & \stackrel{\text{def}}{=} [\![p]\!] \\ [\![ff]\!]_{\mathcal{M}}\rho & \stackrel{\text{def}}{=} \emptyset \\ [\![x]\!]_{\mathcal{M}}\rho & \stackrel{\text{def}}{=} \rho(x) \\ [\![\neg\varphi]\!]_{\mathcal{M}}\rho & \stackrel{\text{def}}{=} S \setminus [\![\varphi]\!]_{\mathcal{M}}\rho \end{array} \qquad \begin{array}{rl} [\![\varphi \supset \psi]\!]_{\mathcal{M}}\rho & \stackrel{\text{def}}{=} (S \setminus [\![\varphi]\!]_{\mathcal{M}}\rho) \cup [\![\psi]\!]_{\mathcal{M}}\rho \\ [\![[a]\,\varphi]\!]_{\mathcal{M}}\rho & \stackrel{\text{def}}{=} \{s \in S \mid \forall r \in [\![a]\!]s : r \in [\![\varphi]\!]_{\mathcal{M}}\rho\} \\ [\![\mu x\varphi]\!]_{\mathcal{M}}\rho & \stackrel{\text{def}}{=} \bigcap\{T \subseteq S \mid [\![\varphi]\!]_{\mathcal{M}}\rho[x \mapsto T] \subseteq T\} \end{array}$$

It is customary to view a formula $\varphi$ with a free variable $x$ as defining a function $\varphi^\rho_x : \mathcal{P}(S) \to \mathcal{P}(S)$, such that for all $U \subseteq S$: $\varphi^\rho_x(U) = [\![\varphi]\!]_{\mathcal{M}}\rho[x \mapsto U]$. The intuitive interpretation of $\mu x\varphi$ is then the *least fixed point* of $\varphi^\rho_x$. The condition on the formation of $\mu x\varphi$ ensures the existence of the lfp:

**Proposition 1** *Let $\varphi$ a formula and $x$ a variable occurring only positively in $\varphi$. Then, in every environment $\rho$, $\varphi^\rho_x$ has both the least and the greatest fixed point. In particular, the lfp of $\varphi^\rho_x$ is $[\![\mu x\varphi]\!]\rho$.*

*Proof.* (Sketch) It is easy to show, by induction on the syntax of $\varphi$, that $\varphi^\rho_x$ is monotone; the result follows from Knaster-Tarski's theorem. □

Notice that this result does not hold if we drop the condition on the formation of $\mu x\varphi$: for instance, the formula $\neg x$ identifies the function $(\neg x)^\rho_x(T) = S \setminus T$, which is not monotone and has no lfp.

In order to have a semantical counterpart of the syntactic notion of "deduction", we introduce a consequence relation for the $\mu$-calculus, which is an extension of the finitary truth CR's of propositional dynamic logic [14].

$$\mathbf{N}C = \boxed{\begin{array}{ll} \Gamma, \varphi \vdash \varphi & \textsc{Raa}\ \dfrac{\Gamma, \neg\varphi \vdash \mathit{ff}}{\Gamma \vdash \varphi} \\ \supset\text{-I}\ \dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \supset \psi} & \supset\text{-E}\ \dfrac{\Gamma \vdash \varphi \supset \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi} \\ \neg\text{-I}\ \dfrac{\Gamma, \varphi \vdash \mathit{ff}}{\Gamma \vdash \neg\varphi} & \mathit{ff}\text{-I}\ \dfrac{\Gamma \vdash \varphi \quad \Gamma \vdash \neg\varphi}{\Gamma \vdash \mathit{ff}} \end{array}}$$

$$\mathbf{N}K = \mathbf{N}C + \boxed{[\cdot]\text{-I}\ \frac{\emptyset \vdash \varphi}{\emptyset \vdash [a]\varphi} \qquad \supset_{[\cdot]}\text{-E}\ \frac{\Gamma \vdash [a](\varphi \supset \psi) \quad \Gamma \vdash [a]\varphi}{\Gamma \vdash [a]\psi}}$$

$$\mathbf{N}\mu K = \mathbf{N}K + \boxed{\mu\text{-I}\ \frac{\Gamma \vdash \varphi[(\mu x.\varphi)/x]}{\Gamma \vdash \mu x.\varphi} \qquad \mu\text{-E}\ \frac{\Gamma \vdash \mu x.\varphi \quad \varphi[\psi/x] \vdash \psi}{\Gamma \vdash \psi}}$$

Figure 1. ND-style systems for classical logic, modal logic and propositional $\mu$-calculus

**Definition 1 (Consequence Relations for $\mu K$)** *Let $\mathcal{M}$ be a model for $\mu K$ and $[\![\cdot]\!]_{\mathcal{M}}$ be the interpretation in $\mathcal{M}$. The* (truth) consequence relation *for $\mu K$ wrt $\mathcal{M}$ is a relation $\models_{\mathcal{M}} \subseteq \mathcal{P}(\Phi) \times \Phi$, defined as follows:* $\Gamma \models_{\mathcal{M}} \varphi \iff \forall \rho. [\![\Gamma]\!]_{\mathcal{M}}\rho \subseteq [\![\varphi]\!]_{\mathcal{M}}\rho$.

*The (absolute)* truth CR *for $\mu K$ is:* $\Gamma \models \varphi \iff \forall \mathcal{M}. \Gamma \models_{\mathcal{M}} \varphi$.

*The* finitary *truth consequence relations is the restriction of $\models$ to finite sets:* $\Gamma \models_f \varphi \iff \exists \Delta \subseteq \Gamma,\ \mathit{finite}. \Delta \models \varphi$.

In the following, for sake of simplicity, we will drop the $_f$, denoting by $\models$ the *finitary* CR $\models_f$.

## 2 A Natural Deduction style system for $\mu K$

Usually, systems for $\mu$-calculus are given in Hilbert style [10,18,?]. Here we present a Natural Deduction style system for $\mu K$, namely $\mathbf{N}\mu K$. This system is composed by a system for classical propositional logic ($\mathbf{N}C$), extended with two rules for the minimal modal logic ($\mathbf{N}K$) and the two the new two rules (introduction and elimination) for the new constructor $\mu$, as presented in Figure 1. These rules are presented in a sequent (Gentzen-like) fashion; this allow us to spell out clearly the side conditions on hypotheses in the $\mu$-E and $[\cdot]$-I rules. Of course, all these rules can be written also in the more customary Natural Deduction style, like the following:

$$\mu\text{-E}\ \frac{\mu x.\varphi \qquad \begin{array}{c}[\varphi[\psi/x]] \\ \vdots \\ \psi\end{array}}{\psi}\ \begin{array}{l}\psi \text{ does not depend on} \\ \text{any other hypothesis} \\ \text{beside } \varphi[\psi/x]\end{array} \qquad\qquad [\cdot]\text{-I}\ \frac{\varphi}{[a]\varphi}\ \begin{array}{l}\varphi \text{ does not depend} \\ \text{on any hypothesis}\end{array}$$

| | |
|---|---|
| PosinP $\dfrac{p \in \Phi_0}{posin(x,p)}$ | NeginP $\dfrac{p \in \Phi_0}{negin(x,p)}$ |
| PosinY $\dfrac{y \in Var}{posin(x,y)}$ | NeginY $\dfrac{y \neq x}{negin(x,y)}$ |
| PosinImp $\dfrac{negin(x,\varphi) \quad posin(x,\psi)}{posin(x,\varphi \supset \psi)}$ | NeginImp $\dfrac{posin(x,\varphi) \quad negin(x,\psi)}{negin(x,\varphi \supset \psi)}$ |
| PosinNeg $\dfrac{negin(x,\varphi)}{posin(x,\neg\varphi)}$ | NeginNeg $\dfrac{posin(x,\varphi)}{negin(x,\neg\varphi)}$ |
| PosinBox $\dfrac{posin(x,\varphi)}{posin(x,[a]\,\varphi)}$ | NeginBox $\dfrac{negin(x,\varphi)}{negin(x,[a]\,\varphi)}$ |
| PosinMu $\dfrac{\text{for } z \neq x : posin(x,\varphi[z/y])}{posin(x,\mu y\varphi)}$ | NeginMu $\dfrac{\text{for } z \neq x : negin(x,\varphi[z/y])}{negin(x,\mu y\varphi)}$ |

Figure 2. The positivity proof system.

Notice that the side conditions of these two rules are very the same: in fact, $\mu$-E can be stated as $\frac{\Gamma\vdash\mu x.\varphi \quad \emptyset\vdash\varphi[\psi/x]\supset\psi}{\Gamma\vdash\psi}$; here, the left subderivation has to depend on no assumptions, like to the necessitation rule $[\cdot]$-I of modal logic.

The rules for $\mu$ have a direct semantic interpretation: the introduction rule states that (the meaning of) $\mu x\varphi$ is a prefixed point of $\varphi^\rho_x$; the elimination rule states that (the meaning of) $\mu x\varphi$ implies, and then "is less than", any prefixed point of $\varphi^\rho_x$. Therefore, these rules state that (the meaning of) $\mu x\varphi$ is the minimum prefixed point, i.e. the least fixed point, of $\varphi^\rho_x$.

The resulting system is then sound and complete with respect to the (finitary) truth consequence relation:

**Theorem 2** *For $\Gamma$ finite set of formulæ, $\varphi$ formula: $\Gamma \vdash \varphi \iff \Gamma \models \varphi$.*

*Proof.* (Sketch) Soundness is proved by showing that each rule is sound. Completeness can be proved as follows. Since $\Gamma$ is finite, $\Gamma \models \varphi \iff \models \bigwedge\Gamma \supset \varphi$. By completeness of Kozen's axiomatization [19,?], there is an Hilbert-style derivation of $\bigwedge\Gamma \supset \varphi$. Therefore, it is sufficient to prove that Kozen's axioms and rules (e.g. those presented in [18]) are derivable in $\mathbf{N}\mu K$. □

Since we aim to encode the $\mu$-calculus in some logical framework, we need to enforce the context-sensitive condition on the formation of formulæ of the form $\mu x\varphi$. That is, we ought to specify in detail the condition of "occurring positive in a formula" for a variable. This notion can be represented by two new judgements on formulæ and variables, *posin* and *negin*, which are derived by means of the rules in Figure 2. Roughly, $posin(x,\varphi)$ holds iff all occurrences of $x$ in $\varphi$ are positively; dually, $negin(x,\varphi)$ holds iff all occurrences of $x$ in $\varphi$ are negative. Notice that if $x$ does not occur in $\varphi$, then it occurs both positively and negatively.

Let us formalize better the meaning of these auxiliary judgements. The notions they capture are the following:

**Definition 2 (Monotonicity and Antimonotonicity)** *Let $\varphi$ be a formula and $x$ a variable. We say that $\varphi$ is* monotone *on $x$ (written $Mon_x(\varphi)$) iff $\forall \mathcal{M}, \forall \rho, \forall U, V \subseteq S$: $U \subseteq V \Longrightarrow \varphi^\rho_x(U) \subseteq \varphi^\rho_x(V)$. We say that $\varphi$ is* antimonotone *on $x$ (written $AntiMon_x(\varphi)$) iff $\forall \mathcal{M}, \forall \rho, \forall U, V \subseteq S$: $U \subseteq V \Longrightarrow \varphi^\rho_x(U) \supseteq \varphi^\rho_x(V)$.*

These notions refer directly to the semantic structures in which formulæ take meaning. The following result proves that the syntactic condition of positivity (respectively, negativity) captures correctly the semantic condition of monotonicity (respectively, antimonotonicity).

**Proposition 3** *For all $\varphi \in \Phi$, $x \in Var$: $\vdash posin(x, \varphi) \Rightarrow Mon_x(\varphi)$ and $\vdash negin(x, \varphi) \Rightarrow AntiMon_x(\varphi)$.*

*Proof.* By simultaneous induction on the syntax of $\varphi$. □

Notice that the converse of Proposition 3 does not hold. Consider e.g. $\varphi \stackrel{\text{def}}{=} (x \supset x)$: clearly, $[\![\varphi]\!]^\rho_{\mathcal{M}} = S$ always, and hence $(x \supset x)^\rho_x$ is both monotone and antimonotone. However, $x$ does not occur only positively nor only negatively in $\varphi$. Correspondingly, we cannot derive $\vdash posin(x, (x \supset x))$ nor $\vdash negin(x, (x \supset x))$. This result can be generalized in the following limitation property:

**Proposition 4** *If $x \in \mathrm{FV}(\varphi)$ occurs both positively and negatively in $\varphi$ then neither $posin(x, \psi)$ nor $negin(x, \psi)$ are derivable.*

*Proof.* (Sketch) By induction on the syntax of $\varphi$. □

However, we can restrict ourselves to only positive formulæ w.l.o.g.: by Lyndon Theorem [6], every monotone formula is equivalent to a positive one.

## 3 The encoding of $\mu$-calculus

In this section we present the encoding of the $\mu$-calculus in the Calculus of Inductive Constructions. We will present both the formalization of the language and of the proof system $\mathbf{N}\mu K$ given in Section 2.

### *3.1 Encoding the language*

The encoding of the language of $\mu$-calculus is quite elaborate. The customary approach, is to define an inductive type, `o:Set`, whose constructors correspond to those of the language of $\mu K$. In order to take full advantage of $\alpha$-conversion and substitution machinery provided by the metalanguage, we adopt the *higher order abstract syntax* [9,7]. In this approach, binding constructors (like $\mu$) are rendered by higher-order term constructors; that is, they take a *function*. The naïve representation of $\mu$, therefore, would be `mu:(o->o)->o`;

however, this solution does not work inside an inductive definition of CIC, because it leads to a non-well-founded definition [4,7,14].

The second problem is the presence of a context-sensitive condition on the applicability of $\mu$: in order to construct a formula of the form $\mu x\varphi$, we have to make sure that $x$ occurs positively in $\varphi$. Inductive types do not support this kind of restriction, since they define only context-free languages [14].

In order to overcome the first problems, we adopt the *bookkeeping via Leibniz equality* technique [14]. We introduce a separate type, `var`, for the identifiers. The rôle played by variables is that of "placeholders" for formulæ: they can be replaced by formulæ in the application of $\mu$-I and $\mu$-E rules. However, we do not introduce any substitution predicate: instead, as we will see in Section 3.2, we will inherit the substitution machinery directly from the metalanguage, i.e., the typed $\lambda$-calculus of CIC.

There are no constructors for type `var`: we only assume that there are infinitely many variables.

```
Parameter var : Set.
Axiom var_nat : (Ex [srj:var->nat](n:nat)(Ex [x:var](srj x)=n)).
```

Then, we define the set of preformulæ of $\mu$-calculus, also those not well formed:

```
Parameter Act : Set.
Inductive o   : Set :=  p : o | ff : o | Not : o -> o
                     | Imp : o -> o -> o
                     | Box : Act -> o -> o
                     | Var : var -> o
                     | mu  : (var->o) -> o.
```

Notice that, the argument of `mu` is a function of type `var->o`. In general, this may arise *exotic terms*, i.e. terms which do not correspond to any preformula of the $\mu$-calculus [7,14]. In our case, this is avoided since `var` is not declared as an inductive set (see Section 11.2 of [14] for further details).

Now, we have to rule out all the non-well-formed formulæ. At the moment, the only way for enforcing in CIC context-sensitive conditions over languages is to define a subtype by means of $\Sigma$-types. As a first step, we formalize the system for positivity/negativity presented in Figure 2, introducing two judgements `posin`, `negin` of type `var->o->Prop`. A careful analysis of the proof system (Figure 2) points out that the derivation of these judgements is completely syntax driven. It is therefore natural to define these judgements as *recursively defined functions*, instead of inductively defined propositions. This is indeed possible, but the rules for the binding operators introduce an implicit quantification over the set of variables *different from the one we are looking for*. This quantification is rendered by assuming a locally new variable ($y$) and that it is different from the variable $x$ (see last cases):

```
Fixpoint posin [x:var;A:o] : Prop :=
   <Prop>Case A of True
                   True
                   [B:o](negin x B)
                   [A1,A2:o](negin x A1)/\(posin x A2)
                   [a:Act][A1:o](posin x A1)
                   [y:var]True
                   [F:var->o](y:var)~(x=y) -> (posin x (F y))
         end
with negin [x:var;A:o] : Prop :=
   <Prop>Case A of True
                   True
                   [B:o](posin x B)
                   [A1,A2:o](posin x A1)/\(negin x A2)
                   [a:Act][A1:o](negin x A1)
                   [y:var]~(x=y)
                   [F:var->o](y:var)~(x=y) -> (negin x (F y))
         end.
```

Therefore, in general a goal (*posin x A*) can be `Simplified` (i.e., by applying the `Simpl` tactic, in Coq) to a conjunction of only three forms of propositions: `True`, negations of equalities or implications from negations of equalities to another conjunction of the same form. These three forms are dealt with simply in the Coq environment, hence proving this kind of goals is a simple and straightforward task.

Then, we can define when a preformula is well formed; namely, when every application of $\mu$ satisfies the positivity condition:

```
Fixpoint iswf [A:o] : Prop :=
   <Prop>Case A of True
                   True
                   [A1:o](iswf A1)
                   [A1:o][A2:o](iswf A1)/\(iswf A2)
                   [a:Act][A1:o](iswf A1)
                   [x:var]True
                   [F:var->o](x:var)
                      ((notin x (mu F))-> (posin x (F x)))
                      /\(iswf (F x))
         end.
```

Hence, each formula of the $\mu$-calculus is represented by a pair preformula-proof of its well-formedness:

```
Record wfo : Set := mkwfo {  prp : o;  cnd : (iswf prp) }.
```

In order to estabilish that our encoding is faithful, we introduce the following notation: for $X = \{x_1, \ldots, x_n\} \subset Var$, let $\Phi_X \stackrel{\text{def}}{=} \{\varphi \mid \mathrm{FV}(\varphi) \subseteq X\}$, $\Gamma_X \stackrel{\text{def}}{=} \mathtt{x_1 : var}, \ldots, \mathtt{x_n : var}$; moreover, let $\mathtt{o}_X \stackrel{\text{def}}{=} \{\mathtt{t} \mid \Gamma_X \vdash \mathtt{t : o},\ \mathtt{t}$ canonical$\}$ and $\mathtt{wfo}_X \stackrel{\text{def}}{=} \{\mathtt{t} \in \mathtt{o}_X \mid \exists \mathtt{d}.\Gamma_X \vdash \mathtt{d : (iswf\ t)}\}$. We can then define the *encoding* functions $\varepsilon_X : \Phi_X \to \mathtt{o}_X$, as follows:

| | |
|---|---|
| $\varepsilon_X(\mathit{ff}) = \mathtt{ff}$ | $\varepsilon_X(\varphi \supset \psi) = \mathtt{(Imp}\ \varepsilon_X(\varphi)\ \varepsilon_X(\psi)\mathtt{)}$ |
| $\varepsilon_X(x) = \mathtt{x}$ | $\varepsilon_X(\mu x\varphi) = \mathtt{(mu\ [x:var]}\varepsilon_{X,x}(\varphi)\mathtt{)}$ |
| $\varepsilon_X(\neg\varphi) = \mathtt{(Not}\ \varepsilon_X(\varphi)\mathtt{)}$ | $\varepsilon_X([a]\varphi) = \mathtt{(Box\ a}\ \varepsilon_X(\varphi)\mathtt{)}$ |

The faithfulness of our encoding is therefore stated in the following theorem:

**Theorem 5** *For $X \subset Var$ finite, the map $\varepsilon_X$ is a compositional bijection between $\Phi_X$ and $\mathtt{wfo}_X$.*

*Proof.* (Sketch) Long but not difficult inductions. First, we prove that `posin`, `negin` adequately represent the positivity/negativity proof system. Due to its structure, it is easy to prove that the type `(posin x A)` is inhabited by at most one canonical form (that is to say, there is at most one way for proving that a preformula is well-formed). Therefore, a preformula $\varphi$ is a formula iff each application of $\mu$ is valid, iff for each application of $\mu$ there exists a (unique) witness of `posin`, iff there exists an inhabitant of $(\mathtt{iswf}\ \varepsilon_X(\varphi))$. □

### *3.2 Encoding the proof system* $\mathbf{N}\mu K$

In the encoding paradigm of Logical Frameworks, a proof system is usually represented by introducing a *proving judgement* over the set of formulæ, like `T:o -> Prop`. A type `(T phi)` should be intended, therefore, as "$\varphi$ is true"; any term which inhabits `(T phi)` is a witness (a proof) that $\varphi$ is true. Each rule is then represented by a type constructor of `T`. A complete discussion of this paradigm, with an example of encoding of $\mathbf{N}C$, can be found in [9,2].

However, in representing the proof system $\mathbf{N}\mu K$, two difficult issues arise: the encoding of proof rules, like $[\cdot]$-I and $\mu$-E, and the substitution of formulæ for variables in rule $\mu$-E. These issues escape the standard encoding paradigm, so we have to accommodate some special technique.

Actually, in the underlying theory of CIC there is no direct way for enforcing on a premise the condition that it is a theorem (i.e. that it depends on no assumptions) or, more generally, that a formula depends only on a given set of assumptions. The solution we adopt exploits again the possibility provided by Logical Frameworks of considering locally quantified premises, i.e. general judgements in the terminology of Martin-Löf; see [3] for a detailed description.

The basic proving judgement is `T:U->o->Prop`, where `U` a set with *no* constructors. Elements of `U` will be called *worlds* for suggestive reasons. Each

"pure" rule (i.e., with no side condition), is parameterized over a generic world, like the following (see Appendix A.2 for the complete listing):

```
Axiom Imp_E : (w:U)(A,B:o)(T w (Imp A B)) -> (T w A) -> (T w B).
```

Therefore, in a given world all the classical rules apply as usual. It should be noticed, however, that we require a locally introduced formula to be well formed. This is the case of $\supset$-I:

```
Axiom Imp_I : (w:U)(A,B:o)(iswf A) ->
                          ((T w A) -> (T w B)) -> (T w (Imp A B)).
```

Indeed, it can be shown that if we allow for non-well formed formulæ in these "negative positions", we get easily an inconsistent derivation.

Proof rules, on the other hand, are distinguished by *local* quantifications of the world parameter, in order to make explicit the dependency between a conclusion and its premises. The $[\cdot]$-I rule is represented as follows:

```
Axiom Box_I: (w:U)(A:o)(a:Act)((w':U)(T w' A))->(T w (Box a A)).
```

The idea behind the use of the extra parameter is that in making an assumption, we are forced to assume the existence of a world, say $w$, and to instantiate the judgement `T` also on $w$. This judgement then appears as an hypothesis on $w$. Hence, deriving as premise a judgement, which is universally quantified with respect to `W`, amounts to establishing the judgement for a generic world on which no assumptions are made, i.e. on no assumptions.

This idea can be suitably generalized to take care of a fixed number of assumptions, like in rule $\mu$-E; here, the dependency between conclusion and assumption is made evident:

```
Axiom mu_E   : (A:o)(w:U)(F:var->o)(iswf A) ->
               ((z:var)(notin z (mu F)) -> (Var z)=A ->
                       (w':U)(T w' (F z)) -> (T w' A))
               -> (T w (mu F)) -> (T w A).
```

This is the most complex rule of the whole system: besides the world parameter technique, it leads us to the second problematic issue of $\mathbf{N}\mu K$, namely the substitution of formulæ for variables, by means of Leibniz equality $=$. A similar, but simpler situation, arises in the encoding of $\mu$-I:

```
Axiom mu_I   : (A:o)(w:U)(F:var->o)
               ((z:var)(notin z (mu F)) -> (Var z)=(mu F)
                       -> (T w (F z)))
               -> (T w (mu F)).
```

The idea is to do not perform substitution immediately; instead it is delayed, until it is actually needed. The binding between the substituted variable `z` and the formula `(mu F)` is kept in the derivation environment by the hypothesis `(Var z)=(mu F)`. Moreover, this hypothesis can be used by the `Rewrite` tactic of Coq, for replacing automatically the variable. Therefore, we do not

need to implement any explicit mechanism for substitution: it is directly inherited from the $\beta$-reduction of the underlying $\lambda$-calculus. For an example of application, see Section A.3.

We need also to locally assume the fact that $z$ does not appear in the formula, i.e. it is *fresh*. This is achieved by the hypothesis `(notin z (mu F))`. The judgement `notin` (and the dual `isin`, see Section A.1) are auxiliary judgements for occur-checking. Roughly, `(notin x A)` holds iff `x` does not occur free in `A`; dually for `isin`. They may be needed in the rest of derivation for inferring well-formness of discharged formulæ in rules RAA, $\supset$-I, $\neg$-I.

The formalization of $\mathbf{N}\mu K$ we have presented is *adequate*, that is, we can derive a property in the system $\mathbf{N}\mu K$ iff we can inhabit the corresponding type in our encoding. This is stated precisely by the following result:

**Theorem 6** *For $X \subset Var$ finite, for $\varphi_1, \ldots, \varphi_n, \varphi \in \Phi_X$: $\varphi_1, \ldots, \varphi_n \vdash_{\mathbf{N}\mu K} \varphi$ iff $\exists \mathtt{t}.\Gamma_X, \mathtt{w} : \mathtt{U}, \mathtt{a_1} : (\mathtt{T\ w}\ \varepsilon_X(\varphi_1)), \ldots, \mathtt{a_n} : (\mathtt{T\ w}\ \varepsilon_X(\varphi_n)) \vdash \mathtt{t} : (\mathtt{T\ w}\ \varepsilon_X(\varphi))$*

*Proof.* ($\Rightarrow$) by induction on derivation; ($\Leftarrow$) by induction on `t`. □

## 4 Conclusions

In this paper we have presented an original encoding of the $\mu$-calculus in type-theory based logical frameworks. We have addressed several problematic issues. First, the extensive and wise use of the higher order abstract syntax frees us from a tedious encoding of the mechanisms involved in the handling of bound names because they are automatically inherited from the metalevel. Secondly, we have faithfully represented the (context-sensitive) language of $\mu$-calculus by formalizing the notion of "well formed formula". Thirdly, the modal nature of the rules of $\mu$-calculus has been rendered, although Logical Frameworks do not support directly modal rules.

The techniques we have adopted can be readily ported to other formalisms featuring similar problematic issues, such as the $\lambda$-calculus, higher-order process calculi, languages defined by context-sensitive grammars, modal logics...

Moreover, our experience confirmed also in dealing with the $\mu$-calculus, is that Logical Frameworks allow to encode faithfully the formal systems under consideration, without imposing on the user of the proof editor the burden of cumbersome encodings. However, nowadays proof editors and Logical Frameworks are still under development; hence, they will benefit from extensive case studies and applications, like the one presented here, which can enlighten weak points and suggest further improvements.

Finally, the encoding presented in this paper could be used as the kernel for a user friendly computer-aided proof environment, in which the user can carry out interactively formal verifications based on the $\mu$-calculus.

## Appendix

## A Coq code

### *A.1 Code of the encoding of the syntax*

```
(* Sets for actions, variables *)
Parameter Act, var : Set.
(* var is at least enumerable *)
Axiom var_nat : (Ex [srj:var->nat](n:nat)(Ex [x:var](srj x)=n)).
Lemma neverempty : (x:var)(Ex [y:var]~(x=y)).
(* proof omitted *)

(* the set of preformulae, also not well formed *)
Inductive o   : Set := p  : o
                     | ff  : o
                     | Not : o -> o
                     | Imp : o -> o -> o
                     | Box : Act -> o -> o
                     | Var : var -> o
                     | mu  : (var->o) -> o.

Fixpoint isin [x:var;A:o] : Prop :=
   <Prop>Case A of False
                   False
                   [B:o](isin x B)
                   [A1,A2:o](isin x A1)\/(isin x A2)
                   [a:Act][B:o](isin x B)
                   [y:var]x=y
                   [F:var->o](y:var)(isin x (F y))
         end.
Fixpoint notin [x:var;A:o] : Prop :=
   <Prop>Case A of True
                   True
                   [B:o](notin x B)
                   [A1,A2:o](notin x A1)/\(notin x A2)
                   [a:Act][B:o](notin x B)
                   [y:var]~(x=y)
                   [F:var->o](y:var)~(x=y) -> (notin x (F y))
         end.

Fixpoint posin [x:var;A:o] : Prop :=
   <Prop>Case A of True
                   True
                   [B:o](negin x B)
```

```
                  [A1,A2:o](negin x A1)/\(posin x A2)
                  [a:Act][A1:o](posin x A1)
                  [y:var]True
                  [F:var->o](y:var)~(x=y) -> (posin x (F y))
         end
with negin [x:var;A:o] : Prop :=
   <Prop>Case A of True
                  True
                  [B:o](posin x B)
                  [A1,A2:o](posin x A1)/\(negin x A2)
                  [a:Act][A1:o](negin x A1)
                  [y:var]~(x=y)
                  [F:var->o](y:var)~(x=y) -> (negin x (F y))
         end.

Fixpoint iswf [A:o] : Prop :=
   <Prop>Case A of True
                  True
                  [A1:o](iswf A1)
                  [A1:o][A2:o](iswf A1)/\(iswf A2)
                  [a:Act][A1:o](iswf A1)
                  [x:var]True
                  [F:var->o](x:var)
                      ((notin x (mu F)) -> (posin x (F x)))
                      /\(iswf (F x))
         end.

(* the set of well formed formulae *)
Record wfo : Set := mkwfo { prp : o;  cnd : (iswf prp) }.

(* separation: if x does not apper in A and y do, then x and y are
 * not the same identifiers - proof omitted *)
Lemma separation : (x,y:var)(A:o)(notin x A) -> (isin y A) -> ~(x=y).

(* an identifier which does not occur,
 * occurs both positively and negatively - proof omitted *)
Lemma notin_posin_negin :
                (x:var)(A:o)(notin x A) -> (posin x A)/\(negin x A).
```

### *A.2 Code of the encoding of the proof system*

```
(* the universe, for the world technique *)
Parameter U : Set.

(* the proving judgement *)
```

```
Parameter T : U -> o -> Prop.

Section Proof_Rules.
Variable A,B: o.
Variable w  : U.

(* proof rules operate also on non-well formed formulae, but for
   ensuring the soundness of the system, we need to require
   well-formness of every discharged formula *)

Axiom ff_I   : (T w A) -> (T w (Not A)) -> (T w ff).
Axiom Not_I  : (iswf A) -> ((T w A) -> (T w ff)) -> (T w (Not A)).
Axiom RAA    : (iswf A) -> ((T w (Not A)) -> (T w ff)) -> (T w A).

Axiom Imp_I  : (iswf A) -> ((T w A) -> (T w B)) -> (T w (Imp A B)).
Axiom Imp_E  : (T w (Imp A B)) -> (T w A) -> (T w B).

Axiom Box_I  : (a:Act) ((w':U)(T w' A)) -> (T w (Box a A)).
Axiom K : (a:Act) (T w (Box a (Imp A B)))
          -> (T w (Box a A)) -> (T w (Box a B)).

Axiom mu_I   : (F:var->o)
             ((z:var)(notin z (mu F)) -> (Var z)=(mu F) -> (T w (F z)))
               -> (T w (mu F)).
Axiom mu_E   : (F:var->o)(iswf A) ->
               ((z:var)(notin z (mu F)) -> (Var z)=A ->
                       (w':U)(T w' (F z)) -> (T w' A))
               -> (T w (mu F)) -> (T w A).
End Proof_Rules.

Lemma ff_E : (A:o)(iswf A) -> (w:U)(T w ff) -> (T w A).
Intros; Apply RAA; Intros; Assumption.
Qed.
```

### *A.3 An example session in* Coq

We will show a complete Coq session, in which we prove that $A \supset \mu x(A \supset x) \vdash \mu x(A \supset x)$. Commands entered by the user are written in *this font*.

```
miculan@maxi:~> coqtop
Welcome to Coq V6.2 (May 1998)
Coq < Require mu.
[Reinterning mu ...done]
```

Let $w$ be a world and $A$ a formula:

```
Coq < Variable w:U. Variable A:o.
w is assumed
```

```
A is assumed
```
We claim the lemma we intend to prove; this leads us in "proof mode":
```
Coq < Lemma simple : (T w (Imp A (mu [x:var](Imp A (Var x))))) ->
                     (T w (mu [x:var](Imp A (Var x)))).
1 subgoal
  ============================
   (T w (Imp A (mu [x:var](Imp A (Var x)))))
    ->(T w (mu [x:var](Imp A (Var x))))

simple < Intros.
1 subgoal
  H : (T w (Imp A (mu [x:var](Imp A (Var x)))))
  ============================
    (T w (mu [x:var](Imp A (Var x))))

simple < Apply mu_I; Intros.
1 subgoal
  H : (T w (Imp A (mu [x:var](Imp A (Var x)))))
  z : var
  H0 : (notin z (mu [x:var](Imp A (Var x))))
  H1 : (Var z)=(mu [x:var](Imp A (Var x)))
  ============================
   (T w (Imp A (Var z)))
```
Now, we need to replace `z` by the corresponding formula, in order to conclude:
```
simple < Rewrite H1.
1 subgoal
  H : (T w (Imp A (mu [x:var](Imp A (Var x)))))
  z : var
  H0 : (notin z (mu [x:var](Imp A (Var x))))
  H1 : (Var z)=(mu [x:var](Imp A (Var x)))
  ============================
   (T w (Imp A (mu [x:var](Imp A (Var x)))))

simple < Apply H.
Subtree proved!

simple < Qed.
(Intros; Apply mu_I; Intros).
Rewrite H1.
Apply H.
simple is defined
```

## References

1. *Handbook of Logic in Computer Science*. Oxford University Press, 1992.
2. A. Avron, F. Honsell, I. A. Mason, and R. Pollack. Using Typed Lambda Calculus to implement formal systems on a machine. *Journal of Automated Reasoning*, 9:309–354, 1992.
3. A. Avron, F. Honsell, M. Miculan, and C. Paravano. Encoding modal logics in Logical Frameworks. *Studia Logica*, 60(1):161–208, Jan. 1998.
4. *The Coq Proof Assistant Reference Manual - Version 6.2*. INRIA, Rocquencourt, May 1998. Available at `ftp://ftp.inria.fr/INRIA/coq/V6.2/doc`.
5. A. Church. A formulation of the simple theory of types. *JSL*, 5:56–68, 1940.
6. G. D'Agostino, M. Hollenberg. Logical questions concerning the $\mu$-calculus: interpolation, Lyndon, and Łoś-Tarski. To be published in the *JSL*, 1999.
7. J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order syntax in Coq. In *Proc. of TLCA'95*. LNCS 902, Springer-Verlag, 1995.
8. G. Gentzen. Investigations into logical deduction. In M. Szabo, ed., *The collected papers of Gerhard Gentzen*, pages 68–131. North Holland, 1969.
9. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.
10. D. Kozen. Results on the propositional $\mu$-calculus. *TCS* 27, 1983.
11. Z. Luo, R. Pollack, and P. Taylor. *How to use LEGO*. Department of Computer Science, University of Edinburgh, Oct. 1989.
12. L. Magnusson and B. Nordström. The ALF proof editor and its proof engine. In *Proc. of TYPES'93*, LNCS 806, pages 213–237. Springer-Verlag, 1994.
13. P. Martin-Löf. On the meaning of the logical constants and the justifications of the logic laws. TR 2, Dipartimento di Matematica, Università di Siena, 1985.
14. M. Miculan. *Encoding Logical Theories of Programs*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, Mar. 1997.
15. M. Miculan. Coq signature for the $\mu$-calculus. Available at `http://www.dimi.uniud.it/~miculan/mucalculus`
16. B. Nordström, K. Petersson, and J. M. Smith. Martin-Löf's type theory. In [1].
17. D. Prawitz. *Natural Deduction*. Almqvist & Wiksell, Stockholm, 1965.
18. C. Stirling. Modal and Temporal Logics. In [1], pages 477–563.
19. I. Walukiewicz. Completeness of Kozen's axiomatisation. In D. Kozen, editor, *Proc. 10th LICS*, pages 14–24, June 1995. IEEE.

**Acknowledgements.** The author is grateful to an anonymous referee for many useful remarks.